SOFTWARE DEVELOPMENT TEAMS IN HIGHER EDUCATION: AN EDUCATOR'S VIEW

David M. Kennedy

Multimedia Education Unit, The University of Melbourne, Australia.
email: d.kennedy@meu.unimelb.edu.au

ABSTRACT

The development of computer-facilitated learning (CFL) for higher education has grown substantially in recent years. The complexity of modern CFL development has resulted in the need to form teams of people with a range of skills – academics for content knowledge, graphic and interface designers for the human computer interface design, programmers for coding, educational designers who focus on pedagogical issues of teaching and learning, and a project manager to coordinate resources, budgets and time lines of the development process. Each member of the project team brings skills and professional knowledge to the group.

In the development of courseware, there are a number of different software engineering models to choose from. Each model is shaped by an underlying set of assumptions about the nature of knowledge acquisition (learning), the most appropriate methodology to achieve a quality outcome, the role of the teacher in the use of the courseware, and a view of the context into which the courseware will be implemented.

In this paper, the pedagogical assumptions that underpin five models of software engineering are examined. The majority of programmers are trained in the Linear Sequential Model (LSM) of software design. This model is closely aligned to a linear instructionist model of instructional design and may create conflicts when a programmer is working with an educational designer who is using a constructivist approach. There are other models of software engineering more closely aligned to constructivist approaches but these are less frequently used in the design of courseware in higher education. These issues will be addressed by reference to a number of courseware projects.

KEY WORDS

Software development, software engineering, constructivist pedagogy, software specifications.

1 INTRODUCTION

The development of academic courses that use new computer-based technologies in higher education has increased markedly in recent times. The use of computer-facilitated learning (CFL), including Web-based courses in academia is increasing because the nature of higher education itself is changing. Students with more diverse academic backgrounds, interests, and motivation now undertake tertiary studies. The times when an early adopter (enthusiast) could design, develop, implement and evaluate CFL courseware as an individual have long passed (if they ever really existed in the first place). The process of developing CFL courseware, Web-based courses and using computer-mediated communications has now become an institutional point of focus rather than merely the domain of enthusiasts and innovators. As CFL software has moved from the fringes of higher education to being core components of course delivery, issues of software quality, student learning outcomes and integration of CFL

modules within the whole curriculum context have become paramount. This has necessitated the formation of multi-disciplinary teams for software development. Ideally, such teams would be composed of individuals with a range of specialist skills now needed to develop a large, complex CFL project (Freeman & Ryan, 1995). These include:

- experience in teaching and educational design (Kennedy & McNaught, 1997);
- video and audio skills;
- programming skills;
- extensive knowledge of the content domain;
- interface and graphical design;
- formative and summative evaluation (Alexander & Hedberg, 1994); and
- project management (Phillips, 1997).

Difficulties arise because no individual has all of these skills and acquiring even a sub-set of them requires considerable investments in time and effort. In any project a model of software engineering will be used, implicitly or explicitly, to support the development of courseware by the team. The questions to be addressed are:

1. What are the underlying pedagogical assumptions that shape models of software engineering?
2. What models of SE best support current models of educational design of multimedia for student learning.

The exploration of possible answers to these two questions may alleviate many potential sources of frustration, misunderstanding and conflict in courseware development teams – delivering a CFL product that enhances student learning in an acceptable timeframe.

2. THE CONSTRUCTIVIST VIEW

Any software developed for educational purposes in order to assist student learning must require students to actively interact with new material in ways which require reflection. It is not sufficient for students to understand an argument or explanation in a detached way. They need to make decisions in their work which show clearly what their own knowledge constructions are. It is my belief that CFL will only assist student learning when the tasks designed are based on the constructivist principles. The principles of a constructivist perspective of teaching and learning can be summarised as follows (McNaught, 1993).

- Students have prior well-formed frameworks of ideas about many of the topics they study in science.
- Learners build up personal, internal conceptual maps as a result of interactive processes between each learner and her or his environment.
- Our frameworks embrace our sociocultural environment as well as our physical environment.
- Learning occurs as an active construction of meaning as a result of reflection on experiences.
- ‘Reflection’ is one of those concepts which deserves to be reflected upon. It does not just mean thinking over an experience, but implies a conscious integration of experience into an existing framework.
- The process of reflection is not purely rational; motivation and interest are essential.

An interactive learning environment based on a constructivist perspective of teaching and learning requires (Kennedy, Fritze, & McTigue, 1997):

- active student engagement in the construction of knowledge;

- the facility to allow for a variety of student inputs;
- provision for an iterative approach to learning; and
- provision for immediate and appropriate feedback.

Developing software with these characteristics requires early, ongoing and meaningful evaluation of each iteration of the courseware with students. Hedberg & Alexander (1994) have developed a model for formative and summative evaluation that addresses interface issues and student learning outcomes in the development of interactive multimedia. They argue that early formative evaluation with the target group will alleviate many potential design problems. This is supported by Moonen & Schoenmaker (1992) who state that the interaction of the user with the program is often very difficult to specify precisely and an early prototype "almost always elicits comments and suggestions for alterations" (p.118). It is not too strong a statement to say that formative evaluation is fundamental in courseware development if a quality product is to be delivered (Burkhardt, 1992).

3. MODELS OF SOFTWARE ENGINEERING (SE)

> Originally ... software engineering was approached as a linear activity in which a series of sequential steps were applied in order to solve problems. Yet, linear approaches to software development run counter to the way in which most systems are actually built. In reality, complex systems evolve iteratively, even incrementally. It is for this reason that a large segment of the software engineering community is moving toward evolutionary models of software development (Pressman, 1997, p. 832).
>
> In an ideal project most participants would be software engineers (Johnston, 1997).

These are two views offered of software engineering. The first is from a very popular textbook on software engineering in its fourth edition, and the second reflects a view experienced by students engaged in their final year of computer science. The first view is sympathetic to the view adopted by a constructivist approach to the educational design of software – the second view is, at best, naive.

Pressman (1997) lists the typical models for software engineering design as:

- Linear Sequential Model (the waterfall model);
- Prototyping Model;
- Rapid Application Development (RAD); and
- Evolutionary software process models – Incremental and Spiral.

These five models are representative of common approaches to software engineering. Other modern models of software engineering (e.g., object-oriented technologies) may offer better solutions in the future for the large courseware projects being developed in higher education (Pressman, 1997). However, the five models have been selected because the expertise and funding to implement them are generally available in higher education. The advantages and disadvantages of each model in an educational context are described in a series of tables (Tables 2, 4, 5, 6, 7). Each table also has a summary of the major aspects of the model. Each of the visual representations of a model has been adapted to reflect terminology familiar to educators in an effort to clarify the model (for educators). For example, the six descriptors in the spiral model (Table 7) have been replaced with expressions more familiar to an educator (Table 1).

Table 1

Descriptors for SE: An Educator's Viewpoint

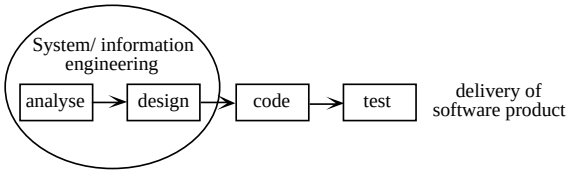
Original descriptor	Educational descriptor
Customer communication	Communication with lecturer: exit and entry point
Planning	Evaluate alternatives, analyse curriculum context, student learning outcomes
Risk analysis	Planning and project management
Engineering	Design of CFL learning environment (engineering) & resource generation
Construction and release	Prototyping of core components/ functional components (coding)
Customer evaluation	Evaluation by students (formative followed by summative)

3.1 THE LINEAR SEQUENTIAL MODEL

The first model, the Linear Sequential Model or 'waterfall model' is the oldest and still the most widely used (Pressman, 1997) – and the most problematic in an educational environment. It is of interest to note that in a typical undergraduate software engineering course (at The University of Melbourne at least), the LSM is still the dominant model used to illustrate the software design approach in the final year software engineering projects (Arnold, Dart, Hassall, & Johnston, 1997). Table 2 provides a summary of this model.

Table 2

The Linear Sequential Model

Model	Advantages	Disadvantages
Linear Sequential Model ('waterfall' model)	1. Provides a template (actually a series of templates) into which methods of analysis, design, coding, testing and maintenance can be placed. 2. It is widely used by software engineers (particularly in business applications) and well understood. 3. It is significantly better than an ad-hoc or haphazard approach to software development and design.	1. Real projects are not developed in a linear fashion. 2. Although an iterative process may be accommodated, in a real project changes tend to cause confusion. 3. In educational projects it is often very difficult to state all requirements explicitly as required by the model. 4. The first working version of the project will not be produced until late in the project time-span. Major problems with students are therefore only discovered after substantial efforts have been invested in programming and design. 5. The linear nature of the model creates situations where delays in one part of the project can 'block' the rest of the project team from proceeding.
Summary Linear Atomistic		

(after Pressman, 1997, p. 31)

3.1.1 Two LSM projects

I am involved in the educational design in both projects. Project A is a large multimedia project focused on producing an interactive CD-ROM. It is well funded (from internal and external sources) and has access to team members with considerable levels of professional expertise and experience. Detailed documentation has been produced. On the surface, (Table 3) Project A appears to have all of the personnel with the requisite skills to succeed – that is, produce a quality product in a specified time. After a year, students are yet to see a prototype. Project B is a small, unfunded Web-based project. The project is part of the requirements for students engaged in an undergraduate degree in Computer Science. In Project B the students are not required to produce a product – the experience of software engineering in a 'real' situation is the primary focus of the course lecturers. However, the students are very committed to achieving a working prototype as well as completing all of the documentation required (on which they are assessed). After seven months there are over 110 pages of highly detailed software requirements for a concept untested in a 'real' educational environment – actual use by academics and students. Table 3 summarises each of the projects.

Despite obvious differences, the two projects are remarkably similar in two respects – they are both using the same model of software engineering, the Linear Sequential Model (LSM) and members of both teams are committed to producing a quality product. The difficulties that arise are that both:

- have yet to produce a viable product for evaluation with students;
- may require substantial revision after user evaluation; and
- will be difficult to alter because a great deal of effort has already been invested in the programming.

Table 3

The Projects

Project	Size of project	Team composition
Project A	A large (potentially) ongoing project in the biological sciences in a major university faculty undergoing considerable change to the fundamental curriculum. In the future, the project may ultimately be too large for one CD-ROM due to the number and complexity of the multimedia elements.	Academic and general staff with the following professional expertise: • graphic design, • video and photography, • image capture, • software programming, • educational knowledge, • project management, • content knowledge, and • discipline specific technical skills.
Project B	A small third year software engineering project for students. The students are 'contracted' to an academic client. The intention is to develop a small Web-based cognitive tool.	The team consists of: • an educationalist, and • software engineering students in their final year of computer science.

3.1.2 The problems with the LSM approach

The major competing factors affecting the completion of the two projects are:

- the programmers are reluctant to undertake substantial programming until all of the documentation is completed; and
- the educators and content experts are attempting to 'second-guess' problems likely to be experienced by students in order to satisfy the documentation requirements of the first group – without the opportunity to generate any formative student evaluation of the software.

This process of 'second-guessing' results in a delay of the final documentation due to frequent changes in the software design (by the content experts) as potential problems (imagined or otherwise) are rectified – to produce a quality product. This has been the situation in both projects (although more so in project A due to greater complexity). The overall results include:

- the delay of any recognisable product;
- frustration and angst for everyone involved; and
- a significant amount of coding will be done before any evaluation with either target group.

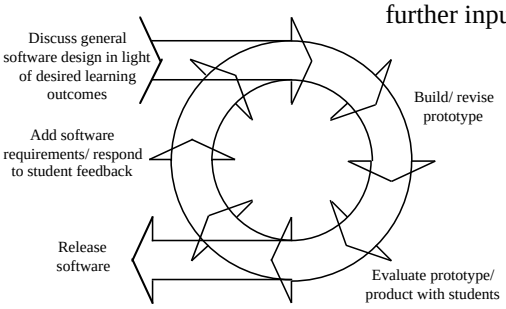
The major problem with the extensive documentation (e.g., Software Quality Assurance Plan (SQAP), Software Requirements Specification (SRS), Software Design Document (SDD) and the Test Plan (TP)) required by the waterfall model of software engineering is that in an educational project in which limited or no evaluation with students has been carried out 'the specification is difficult to test in any meaningful way, so inconsistencies or omissions may pass unnoticed (Pressman, 1997, p. 293)'. The educator is forced to develop software with minimal input from the end-user. Equally, it is unrealistic for programmers to have to cope with frequent requests for modifications to the interface, functionality, or content of a project from the content expert. It is clear from the experience of the two projects that the LSM is not well suited for software development in higher education – from a programmer's or an educator's perspective. An iterative, modular or component-based approach with significant student input would be more effective.

3.2 THE PROTOTYPING MODEL

Table 4 provides an overview of the Prototyping model. Models of software engineering which instigate earlier involvement with students (ie., support early formative evaluation) are more likely to foster quality outcomes, and satisfy the requirements of programmer and educator. This view is not universally supported. Alan Cooper (1994) the designer of Visual Basic claims that 'prototyping is programming and it is harder than concrete to change'. He claims programmers don't like to alter working code and the issues discussed in Table 2 (keeping inappropriate programming structures) become problematic.

Table 4

The Prototyping Model

Model	Advantages	Disadvantages
Prototyping Model	1. It is very suitable for projects in which the specifications (e.g., the form of the software that most enhances student learning) are not yet fully understood. 2. The prototype becomes a mechanism for the expression of the software requirements to be incorporated in the final product. 3. The potential users (students) have the opportunity to influence the form the final product because evaluation is initiated early in the design and development of the software.	1. There is a danger of raising expectations. The time and effort required to develop a stable, robust final form of the software may not be well understood by the academic developer. 2. The software 'fudges' used to get a basic form of the prototype working quickly may remain in the final product – compromising the integrity of the software. 3. There may be an unreasonable expectation from the academic that the final product may be better than can be delivered within budgetary and time constraints. 4. There may be a loss of team enthusiasm due to the mistaken belief that the project is almost complete and there is little need for further input into the project.
Summary Early prototupe Students involved Prototype informs design		(after Pressman, 1997, p. 33)

However, Gunn (1995, p. 188) reported a conflicting view that highlights a major difference between the business/ industrial environments in which most SE models were developed, and higher education. In this study, what was initially a throw-away prototype was viewed so positively by students and experts in the learning environment, it became the basis for further development for specific use in tutorials. Clearly, a model with a prototyping component can provide very useable software, and guide the design process in a meaningful way. However, there needs to be some care in selecting projects which use this methodology.

3.3 THE RAPID APPLICATIONS DEVELOPMENT MODEL

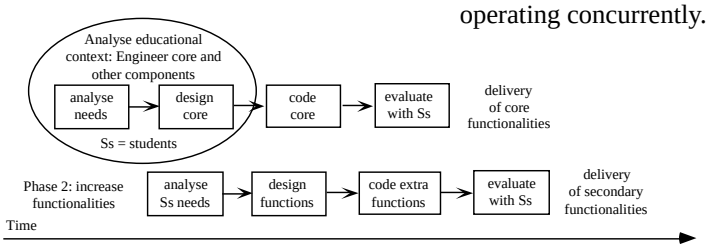
Rushby (1997, p. 18) advocates the Rapid Applications Development (RAD) model of software development as a more efficient method of software development in contrast to the traditional waterfall process.

> A major benefit is the early and continual involvement of end users ... (using the RAD approach) means that users have a real say in the evolution of the package through its prototypes and are empowered to suggest significant changes to structure, content and functionality.

Table 5 examines the RAD model. The RAD model is being used very successfully at the Royal Melbourne Institute of Technology in the Faculty of Engineering to develop a series of templates for courseware development (Kennedy, McNaught, & Nicolettou, 1998, In this volume).

Table 5

The Rapid Applications Development Applications Development Model

Model	Advantages	Disadvantages
Rapid Applications Development (RAD) model	1. The RAD model produces demonstrable products quickly (typically 60 to 90 days). 2. The design and development of software proceeds incrementally – each component is evaluated with end users before being added to the next iteration. 3. There is early involvement of the end-users (students).	1. This model requires an approach derived from re-useable components. 2. The scope of the project and the specific requirements must be constrained. 3. The model has been used most for information systems in business applications. 4. In larger projects there may be a need for a set of development teams operating concurrently.
Summary Incremental End users involved Modular		

(after Pressman, 1997, p. 38)

3.4 THE EVOLUTIONARY MODELS

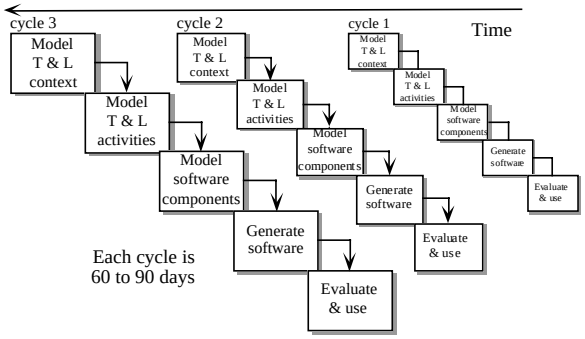
The evolutionary models of software engineering (Incremental and Spiral, Table 6 and Table 7 respectively) have a framework that is iterative, involve the target group early, and have the potential to be used in situations requiring rapid development. Both models represent developmental scenarios more representative of the ‘real’ world. A model of software development derived from the Incremental model has been used in the development of two online tools, Learning Evaluation Online (LEO) (Kennedy & Ip, 1998) and the Text Analysis Object (Kennedy, Ip, Eizenberg, & Adams, 1998, In this volume). In both the LEO and the TAO projects, the core functionalities were devised using a scenario-based approach (Carroll, 1995). Initial formative evaluation with potential users (students and academic staff) was achieved using focus groups (students) and individuals (staff) with either screen mock-ups or descriptions of functionalities and what-would-you-do-if scenarios.

Table 6

The Incremental Model

Model	Advantages	Disadvantages
Incremental model	1. Combines the linear organisational components of the LSM with the iterative philosophy of prototyping. 2. It can produce a 'useable' product with limited functionality quickly (the first and subsequent iterations add more functionality). 3. It focuses on delivering an operational product, with each iteration. 4. Feedback and useability testing can suggest improvements in functionality which add to, or modify the core components. 5. There is early and on-going involvement with the target group (students). 6. Increments can be planned to take advantage of new hardware or software becoming available (e.g., faster CD-ROM drives, of better video compression software).	1. It may be difficult to pre-empt what functional components will be needed in the future. Integration into a single coherent piece of software may be difficult. 2. The scope of the project and the specific requirements must be constrained. 3. The model is most suitable for systems that can be delivered as a series of inter-operable components.

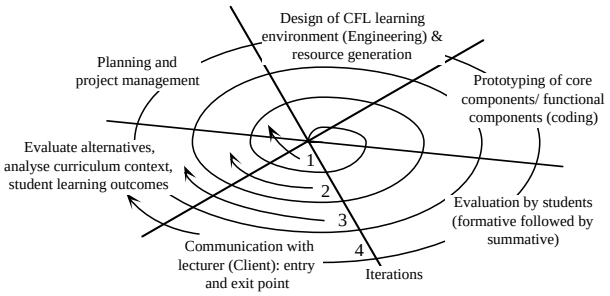
Summary
Modular
Iterative
Ordered
Involved with students
Early evaluation



(after Pressman, 1997, p. 36)

Table 7

The Spiral Model

Model	Advantages	Disadvantages
Spiral model	1. Combines the iterative nature of prototyping with the technical and systematic components of the LSM. 2. The model provides the potential for rapid development of a first prototype (a paper or a software version) which can then be evaluated by the target group (students). 3. The (student) user has an opportunity for early evaluation of the software with the potential to provide valuable input to improve the quality of the software. 4. The model supports projects that are intended to be implemented for a long time and require ongoing upgrading and adaptation to changing situations (as in changes to higher education curricula). 5. The model is more suitable for large projects – the iterative model employed is more realistic of real-world applications.	1. The model is relatively new and may not be well understood or explainable to members of the software development team. The software engineers may find the model lacking in detail (levels of documentation), and the academic members may not view the process as able to deliver useable results in an acceptable time span. 2. The assessment of risk factors that prevent the project from delivering a quality product in an acceptable time is crucial – the project manager's knowledge and experience play a critical role in determining potential problems early in the development process.
Summary Modular Ordered Involved with students Early evaluation		

(after Pressman, 1997, p. 40)

The functionalities (e.g., authoring interface, feedback to students, student interface) for each tool were then described and a prototype with core functionalities constructed.

A second round of formative evaluation occurred, suggesting improvements to the interface design and construction of additional functionalities in the next cycle of development. Both projects have had very positive formative evaluations (in both rounds) and the time-line to deliver the first prototype of each was measured in weeks. The early feedback from the users has been vital in defining the core functionalities of both tools – while avoiding potentially lengthy delays trying to second-guess the user perceptions.

4. DESIGNING CFL

> (As instructional designers) …we also need to recognise that we are seriously behind our cognate fields in the incorporation of cognitive-based instructional models, the change from 'waterfall' design methodology to a rapid application development (RAD)-based, short cycle time, iterative design methodology, and quality management.
>
> (Forshay, 1997)

The paradigms that people adopt for the development of educational multimedia reflect prior knowledge and experience, the manner in which they were taught, and implicit (or explicit) models of teaching and learning she or he has experienced in their own educational undertakings (Bain & McNaught, 1996). The adage that 'people teach as they were taught' may be extended to 'people design educational multimedia based upon their experiences (and perceptions) of teaching and learning'. In the literature of interactive multimedia development there has been strong support for the use of a constructivist perspective on teaching and learning for more effective instructional design in an educational context (Duffy & Jonassen, 1991). A constructivist approach to software design requires formative evaluation in the design and development phases. The formative evaluation may involve peer review, walk-through of a rapid prototype, observation of target group using the software, user-tracking, and interviews (individually or in focus groups) with potential users (after Hedberg & Alexander, 1994). The student evaluations inform all aspects of the design including the user interface, navigation, and how the software functionality supports student learning.

In software engineering the development of CFL is guided by a number of key stages which attempt to make the software requirements unambiguous, consistent and complete. Each key stage (in some models) is associated with extensive documentation intended to minimise potential errors and maximise quality. On the surface this sounds very much like what the educational designer wishes to accomplish – a quality product which enhances learning. However, the underlying assumptions about knowledge and process that guide the five models may be divided into two groups (Table 8). Table 8 is adapted from Reeves (1992b) pedagogical dimensions of interactive learning systems to include thinking tools, methodologies, and developments in software engineering models over time. The allocation of the particular software engineering model is derived from reference to the literature and experience within the four projects.

Table 8

Dimensions of Software Engineering Models (after Reeves, 1992a)

Dimension	Early models of SE	Later models of SE
Paradigm	Instructionist	Constructivist
Methodology	Atomistic (divide-and-conquer)	Holistic (curriculum context)
Underlying psychology	Behavouralist	Constructivist
Thinking tools	(Flow diagrams, branched linear)	Concept mapping (hierarchical, interconnected layers)
Experiential value	Concrete	Abstract
Role of teacher	Teacher proof	Facilitator
Value of errors	Error free	Experience
Model of Software Engineering	Linear Sequential Model (waterfall model)	Protoyping, Rapid Application Development, Incremental, and Spiral.

The models of software engineering that best support current approaches to educational design of courseware are not the most commonly used (e.g., Project B) by students engaging in learning computer science in higher education (clearly, this refers to courses at this institution).

By training, education, and experience many programmers are most familiar with the LSM or 'waterfall' model. In the LSM approach user testing occurs late in the lifetime of the project. The use of prototypes is confined to debugging and interface issues rather than user perceptions

and formative evaluation as a mechanism for guiding design. There is an assumption that it is possible to specify all elements, functionalities and interface issues in a particular piece of courseware, prior to the commencement of programming. In an educational setting, this is often not possible. Equally, an ad-hoc approach is also to be avoided. The documentation should guide the development of courseware rather than stifle it. Input should be sought from all stakeholders, academics, students, and educational designers prior to and during the development of courseware. The examples provided in this paper of the use of the RAD and Incremental models provide evidence of alternative approaches to developing courseware in higher education – incrementally, with significant user input, and delivering quality outcomes.

Understanding the underlying assumptions of the model of software design chosen for a project and communicating its underlying strengths and weaknesses to the design team early in the life of the project will enhance the design process. Potential conflicts may be alleviated and the strengths of the model used to promote quality outcomes. The project leader, manager, or educational designer must take the initiative to communicate what issues have the potential to cause conflict, the particular responsibilities of each member of the team, and why a software engineering model will be used for a particular project.

The LSM approach has little to recommend it. The Prototyping, RAD, Incremental and Spiral models all offer a pedagogical perspective of software design which are congruent with a constructivist perspective of teaching and learning, and software development. These models offer a framework to develop quality educational software on time and on budget, minimise frustration and compromise, and facilitate meaningful input from all of the stakeholders (academic staff, technical staff and students).

5. REFERENCES

Alexander, S., & Hedberg, J. (1994). Evaluating technology–based learning: Which model? In K. Beattie, C. McNaught, & S. Wills (Eds.), *Interactive multimedia in university education: Designing for change in teaching and learning* (Vol. A 59, pp. 233-244). Amsterdam: Elsevier B. V. (North Holland).

Arnold, T., Dart, P., Hassall, M., & Johnston, L. (1997). *Software Engineering Project Manual: 433-340 Software Engineering Project.* The University of Melbourne: Department of Computer Science.

Bain, J., & McNaught, C. (1996). Academics' educational conceptions and the design and impact of computer software in higher education. In C. McBeath & R. Atkinson (Eds.), *The learning superhighway. New world? New worries?* Proceedings of the Third International Interactive Multimedia Symposium, (pp. 56-59). Perth, Western Australia: Promaco Conventions Pty Ltd.

Burkhardt, H. (1992). Classroom observation in courseware development. *International Journal of Educational Research,* 17, 87-98.

Carroll, J., M. (1995). Introduction: The Scenario perspective on system development. In J. M. Carroll (Ed.), *Scenario-Based Design: Envisioning work and technology in system development* (pp. 1-18). New York: John Wiley and Sons, Inc.

Cooper, A. (1994). *The Perils of Prototyping*. Available: http://www.cooper.com/articles/vbpj_perils_of_prototyping.html [1998, August 2].

Duffy, T. M., & Jonassen, D. H. (1991). Constructivism: New implications for instructional technology? *Educational Technology,* 31: 5, 7-12.

Forshay, R. (1997, 23 Feb). *CBT's first generation: If we're so smart, why ain't we rich?*, [Discussion paper]. Instructional Technology Forum. Available: http://itech1.coe.uga.edu/ITForum/home.html [1997, 23 Feb].

Freeman, H., & Ryan, S. (1995). Supporting the process of courseware development: From concept to delivery. In H. Maurer (Ed.). *ED–MEDIA 95.* Proceedings of the World Conference on Educational Multimedia and Hypermedia, (pp. 223-228). Graz, Austria: Association for the Advancement of Computing in Education.

Gunn, C. (1995). Useability and beyond : Evaluating educational effectiveness of computer-based learning. In G. Gibbs (Ed.), *Improving student learning through assessment and evaluation* (pp. 168-190). Oxford, England: Oxford Centre for Staff Development.

Hedberg, J., & Alexander, S. (1994). Implementation and evaluation: The forgotten end of interactive multimedia development. In M. Ryan (Ed.). *APITITE 94.* Proceedings of the Asia Pacific Information Technology in Training and Education Conference and Exhibition. Brisbane: APITITE 94 Council.

Johnston, L. (1997). *Software Engineering 3A, 433-341: First Semester Handbook and Notes*. The University of Melbourne: Department of Computer Science.

Kennedy, D. M., Fritze, P., & McTigue, P. (1997). An interactive graphing tool: The meeting of pedagogy and technology. In R. Kevill, R. Oliver, & R. Phillips (Eds), *What works and why, ASCILITE'97.* Proceedings of the Australian Society for Computers in Learning in Tertiary Education Annual Conference, (pp. 331-337). Curtin University of Technology, Perth: Academic Computing Services.

Kennedy, D. M., & Ip, A. (1998). Learning Evaluation On-line (LEO): A customisable Web-based evaluation tool. In C. Alvegård (Ed.). *CALISCE '98.* Proceedings of the Fourth International Conference on Computer-aided Learning and Instruction in Science and Engineering, (pp. 255-262). Chalmers University of Technology, Göteborg, Sweden: Chalmers University of Technology.

Kennedy, D. M., Ip, A., Eizenberg, N., & Adams, C. (1998, In this volume). The Text Analysis Object (TAO): Engaging students in active learning on the web. In R. M. Corderoy (Ed.). *FlexibilITy: The next wave.* Proceedings of the Australian Society for Computers in Learning in Tertiary Education Annual Conference. University of Wollongong, Australia.

Kennedy, D. M., & McNaught, C. (1997). Design elements for interactive multimedia. *Australian Journal of Educational Technology,* 13: 1, 1-22.

Kennedy, P., McNaught, C., & Nicolettou, A. (1998, In this volume). Flexible and AGILE. In R. M. Corderoy (Ed.). *FlexibilITy: The next wave?* Proceedings of the Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE). University of Wollongong, Australia.

McNaught, C. (1993). Which science? Which language?, *Science and Mathematics Education Papers 1993* (pp. 148-171). Hamilton: Centre for Science and Mathematics Education Research: University of Waikato.

Moonen, J., & Schoenmaker, J. (1992). Evolution of courseware development methodology: Recent issues. *International Journal of Educational Research,* 17, 109-121.

Phillips, R., A. (1997). *A developer's handbook to interactive multimedia: A practical guide for educational applications*. London: Kogan Page.

Pressman, R. S. (1997). *Software engineering: A practitioner's approach*. (Fourth ed.). New York: McGraw-Hill Companies Inc.

Reeves, T. (1992a). Effective dimensions of interactive learning systems. In A. Holzl & D. Robb (Eds.), *Finding the future: ITTE '92.* Proceedings of the Information Technology for Training and Education Conference, (pp. 99-113). Lucia, Brisbane: University of Queensland.

Reeves, T. C. (1992b). Evaluation of interactive multimedia. *Educational Technology,* 32:5, 47-53.

Rushby, N. J. (1997). Quality criteria for multimedia. *Association for Learning Technology Journal,* 5: 2, 18-30.